Pracownia Technik Obliczeniowych

Debugowanie i testy

Paweł Daniluk

Wydział Fizyki

Wiosna 2016

Jak szukać błędów w programach?

- komunikaty diagnostyczne
- asercje
- debugowanie *post-mortem*
- debugowanie *step-by-step*

Błędy często występują “losowo”.

Przykład

profile_test.py

```
#!/usr/bin/python

import random

for i in xrange(10000):
    a=random.randint(1,10)
    b=random.randint(1,10)
    c=random.randint(1,10)

    c/=a-b
```

Co jest przyczyną wyjątku? Jakie wartości a, b i c go powodują?

Wystarczy uruchomić `ipython` z opcją `-i`.

Przykład

```
#!/usr/bin/python

import random
import math

def natnumber():
    res=random.randint(0,1000)
    if(random.randint(0,1000)>998):
        res=-res
    return res

tab=[natnumber() for _ in xrange(200)]

sum=0
for i in tab:
    sum+=math.sqrt(i)

print sum
```

Przykład

1. Debugowanie *post-mortem* nie daje odpowiedzi, skąd biorą się liczby ujemne.
2. Można uruchomić program pod kontrolą `ipdb` i przejść po wszystkich instrukcjach.
3. Breakpoint: `break 14`
4. Breakpoint warunkowy: `break 10, res<0`

Podłączanie ipdb do działającego programu

```
import ipdb; ipdb.set_trace()
```

Testy jednostkowe

https://docs.python.org/2/library/unittest.html

Testy jednostkowe

Jak konstruować testy jednostkowe?

- wymagania, które musi spełniać program (np. tożsamości algebraiczne)
- przypadki szczególne, które powinny być obsłużone
- błędy i problemy wykryte podczas eksploatacji

Zadanie 1

Wypróbuj przykład z dokumentacji do modułu unittest.

Zadanie 2

Zaimplementuj klasę Complex realizującą podstawowe operacje na liczbach zespolonych oraz testy jednostkowe do niej.